Universidad Carlos III de Madrid

Universidad Carlos III de Madrid

Trabajo Fin de Grado

Desarrollo de un programa para maquetación de exámenes tipo test en formato PDF

Autor: Álvaro Ibáñez Aguirre
Tutor: Jesús Arias Fisteus

Grado en Ingeniería Telemática

Leganés, a 24 de septiembre de 2014

Agradecimientos

En primer lugar a Jesús, por haberme ofrecido la oportunidad de realizar el proyecto y por la paciencia y ayuda prestada.

A mis familiares y amigos que siempre se han preocupado por mí, desde el primer momento en que pise la universidad, hasta este momento.

A mis compañeros de la carrera, porque sin ellos superar este tramo de mi vida habría sido imposible. Quiero hacer una mención especial a Alberto Rodríguez, Adrián Recio y Javier Barón, por haber sido mis escuderos en esta aventura. No me puedo olvidar aun así de gente como Richi, José, Antelo, Lucero, Alguacil, Arellano y un largo etcétera que me han ayudado a prosperar.

A mis padres, por su infinito apoyo en las buenos y en los malos momentos. Porque si llego a ser una persona de provecho algún día se deberá a su tozudez y perseverancia conmigo.

A mi abuela Amparo, que sé que le habría gustado verme convertido en ingeniero.

Y por último a mi amigo Nacho, que por capricho del destino no hemos podido acabar la aventura que iniciamos juntos, este proyecto va por ambos.

Gracias a todos, de corazón.

Resumen

Este documento tiene como objetivo describir los procesos de planteamiento, diseño y desarrollo que se han llevado a cabo para la realización de un programa de maquetación de documentos en formato PDF basado en Python.

Este proyecto se centrará en desarrollar una alternativa para la maquetación de exámenes de tipo test en un proyecto ya existente: Eyegrade.

Para la realización de este proyecto se hará uso de soluciones de software libre, siendo Python el lenguaje elegido en el que se desarrollará el programa, haciendo uso de la librería ReportLab para la maquetación y generación de dichos documentos, en alternativa a la solución actualmente existente en LaTeX.

Para ello será necesario un estudio previo de las posibles alternativas en Python para la maquetación y generación de archivos PDF, así como un análisis de la solución actual para integrar nuestro programa dentro de la lógica ya establecida.

Por último, se realizarán las pruebas necesarias para comprobar que el programa responde como se espera y que representa una alternativa real y funcional al modelo ya existente.

Abstract

This document aims to describe the processes of approach, design and development which have been carried out for the realization of a layout program for documents on PDF format based on Python.

This project will focus on developing an alternative for the layout of multiple choice tests in an existing project: Eyegrade.

For the execution of this project we will make use of free software solutions, choosing Python as the language in which this program will run and using the ReportLab library for the layout and breed of such documents, alternatively to the actual existing solution from LaTeX.

To be able to do this it will be necessary a preliminary study of the possible alternatives in Python for the layout and generation of PDF files, as well as an analysis of the current solution to integrate our program into an already existing logic.

Finally, three tests will be carried out to ensure that the program responds as expected and that it represents a real and functional alternative to the existing model.

Extended summary

Throughout this report the user will be presented with a Final Project (TFG), which attempts to develop an alternative layout and generating program of PDF files for multiple choice tests based on the Python language, to an already existing and integrated to a project program.

The application will seek to provide a functionally equivalent solution to the existing one, trying to integrate into the established design with minimal impact on the execution logic.

Nowadays there are many tools for automatic correction of multiple choice exams. Within this framework we find the Eyegrade project, developed in Python. This tool provides an open source solution for the correction of the tests, being required only the use of a document which follows structure specifications imposed by developers.

Besides the basic functionality of automatic qualification tests, Eyegrade provides a tool for generating tests in PDF format, which currently uses the LaTeX system for layout.

The main motivation for this project emerges from the idea of creating an alternative that does not require external software like LaTeX for the Eyegrade project; this is why we have sought solutions that were already present in the form of external libraries from Python.

The main objective to be achieved with the completion of this project is to develop a program that throughout the use of Python and the ReportLab library, may pose a layout of PDF documents equivalent to the existing project in Eyegrade (explained in detail in the later sections of this memory).

Apart from this, other objectives to be achieved during the preparation of this Final Project are:

- Learn how to use Python's associated libraries such as ReportLab
- Create a real alternative for those users who are not familiar with LaTeX.
- Studying the structure of an alien project and be able to integrate my solution without altering it.
- Learning how to document and argue the completion of a project.

Eyegrade emerges as an application with the primary objective of providing a low cost solution to qualify multiple-choice questions (MCQ) tests, using a webcam to rate those years. The program is free software, and is under the terms of version 3 or later of the GNU GPL.

There are many applications that offer a similar service Eyegrade posed as but the main difference regarding Eyegrade and the others qualification softwares is the ability to use almost any low-end webcam that can be affordable for everyone, in contrast to other solutions that are based on the use of more advanced scanners for visual recognition of the results.

Currently the project is still in alpha stage, but is fully functional and, since 2010, has been used in several courses at the Carlos III University of Madrid.

Eyegrade makes use of XML files and LaTeX templates to allow the user the document layout as he wants. The XML file must include at least the information of the various questions that consist of: a text corresponding to the question itself and the responses indicating which is correct and which is not, also being able to provide data in the file as: the name of the subject, the course that corresponds to that subject, title examination, test date, or duration. These data are not required to insert in the XML, as there is the possibility of passing those arguments through console when running the program.

The whole design of the structure of the document is done through a LaTeX file (.tex) format, which will serve as a template to distribute the various elements of the exam in the pages of the PDF. The most complex of all will be the cover, as it will have both dynamic elements that vary depending on the data entered the program, and static elements such notices, and regulations for the examination, that can only be modified in the LaTeX template.

As for grading examinations, the program creates an .eye file, which contains the necessary data so Eyegrade can grade them. This file contains metadata about exam such as the dimensions of the tables, the number of digits in the ID table, the sample tests, the correct answer to each question, and the permutations that have been made to shuffle the questions and choices. Relying on data from that file, and the database of students, we will proceed to grade examinations focusing them with the webcam for subsequent export of ratings and statistics to a CSV file.

The software will detect the written options as well as the ID of the student, thus entering mode Eyegrade review. For cases in which the program does not detect the correct student ID box, or does not detect the answer tables, there are solutions arranged in two ways to address these situations:

- If the ID is the only thing that has not been detected once in the exam review mode, the user is allowed to select the student from the class list, proposing the most similar to the number recognized by the OCR.

- Failure to detect any of the tables, there is a manual mode where the user must indicate the corners of the first table of responses and the two top corners of the second, to aid recognition thereof.

After comparing similar technologies, it was decided to use ReportLab, which is a library for generating PDF files. It is designed to meet the needs of creators and web developers and professional designers who want to create or automate complex documents quickly and easily.

The library implements a clever and flexible design that called Platypus, you can build documents from elements such as headlines, paragraphs, fonts, tables, graphs, etc. This function closely resembles the ReportLab use LaTeX because it simplifies the way to place the elements, although still control something more. Also, added to the 14 standard PostScript fonts full supports for Type-1 and Asian sources.

Besides the basic functionality and Platypus, ReportLab makes available to users the call RML, which based on language similar to XML tags, the layout allows easy document. The downside of this feature is that it is a paid version unlike other tools.

Furthermore, ReportLab is under the BSD license, which allows almost anything with the code, including use in a commercial package. As part of the development is to be performed in ReportLab which is to be integrated in a project under the GPL, no compatibility issue

The requirements that have to consider our design to recreate the operation of the solution-based LaTeX and attempt to resemble as much as possible the execution logic already established, are:

- It will be necessary to preserve the structure of the input XML file, and as far as possible, the data entered into it.

- Should be sought and include the same font using LaTeX, thus maintain a uniform style and no significant difference between the two solutions.

- The font used for code blocks has to be a fixed-width font.

- Must be possible to include images, if possible as vector graphics (such as EPS formats, PDF, etc.).

- Test models that can be generated will be between the A-H values, accounting for each model a different mixing order of the questions and options. You will also be possible to generate a model 0, which will keep the order of the questions exactly as they are provided in the XML file, having to do the same with the answers, except that the correct answer is always the first choice.

- Each model has to carry associated four black squares to be placed under tables of responses, and serve to identify the model of exam.
  The first three will be used to recognize the pattern, and the fourth will be redundant with the other three. The square will have two positions: up and down; and since there are three squares to identify the model with two possible positions, resulting in eight possible combinations, there are currently only supported models A through H (the first eight letters).

- When generating the part of the questions, it is mandatory to cover all possible cases that may occur texts of questions and options:
  - The texts of the questions may be accompanied by: a block of code whose indentation must be respected to preserve its essence; or an image.
  - Options may be formed by text, code or images; but never more than one type at a time.

- The texts of both the questions and the answers may currently have marks LaTeX to make modifications to the font styles. It is important to assess the possibility of developing such functionality.

ReportLab doesn't allow the use of vector graphics in your documents, unlike LaTeX. ReportLab natively supports only the JPEG format.

The solution that has been proposed for this problem has been replaced by vector graphics images in PNG format, which will be supported by the inclusion of the PIL library, in charge of handling images in Python.

The guidelines established for the development of the design are as follows:

- Reuse most of possible code: Since the important in this project is the layout of the document, it will try to preserve as far as possible: the boot code from console, parsing arguments and subsequent use, generation under review and further handling for use in the call document generation.

- Using ReportLab tool Platypus: Platypus is a library of high ReportLab layout that allows creating complex documents with little effort. It has been used due to the simplification that arises when modifying styles of the different elements, and that although ReportLab allows us to manage and locate the various components of our work, for a job like ours in which repeat actions font change, creating tables, etc., Platypus is a much better option.

- Perform a single template: While it may not seem like the best option for the mess that can occur in the source class has decided to join the entire design in a single class layout, this design will be improved in the future but allows a stranger to locate all the elements used in the same class in a simple user.

Making use of the two existing classes in Eyegrade, we have generated a third class that will act as a template and also will be generating the PDF document using ReportLab and Platypus.

After completing the project, we can ensure that:

- The application development is a real and practical alternative to the existing one, providing a different approach to document layout.
- The mentioned requirements are fulfilled except for one, the implementation of language marks in the texts of questions and choices, this decision was taken because of time and not being a critical point for system operation was decided to leave as a future work line.
- The project leaves marked guidelines for future programmers leaving four possible lines of research and development that would enhance the software.

Personally, the project has helped us to acquire new skills and knowledge to develop applications in Python, and PDF layout programs.

With the above, we can conclude that the project expectations have been met in a satisfactory manner.

Tabla de contenidos

Índice de ilustraciones

Índice de tablas

1. Introduction

Throughout this report the user will be presented with a Final Project (TFG), which attempts to develop an alternative layout and generating program of PDF files for multiple choice tests based on the Python language, to an already existing and integrated to a project program.

The application will seek to provide a functionally equivalent solution to the existing one, trying to integrate into the established design with minimal impact on the execution logic.

1.1. Motivation

Nowadays there are many tools for automatic correction of multiple choice exams. Within this framework we find the Eyegrade project, developed in Python. This tool provides an open source solution for the correction of the tests, being required only the use of a document which follows structure specifications imposed by developers.

Besides the basic functionality of automatic qualification tests, Eyegrade provides a tool for generating tests in PDF format, which currently uses the LaTeX system for layout.

The main motivation for this project emerges from the idea of creating an alternative that does not require external software like LaTeX for the Eyegrade project; this is why we have sought solutions that were already present in the form of external libraries from Python.

With this, in addition to ease the access to the layout tool, also the impact on learning time could be reduced because, since everything is related to Python and its libraries, any user who knows the language, will be able to understand the functioning of the application without having to possess the skills on the ReportLab library (and in the case of users who already know how Python handles PDF documents, will be even less).

On the other hand, ReportLab is a lower level tool. With LaTeX, the user is restricted to use macros provided by the system, whereas in the first one the user has finer control over the output document.

One of the incentives which moved me to develop this solution was because of the possibility of extending my knowledge in the field of learning a new programming

language, since I was completely unaware of the operation and Python syntax; as well as having the chance to interact with one of their extensive libraries.

1.2. Goals

The main objective to be achieved with the completion of this project is to develop a program that throughout the use of Python and the ReportLab library, may pose a layout of PDF documents equivalent to the existing project in Eyegrade (explained in detail in the later sections of this memory).

Taking advantage of the knowledge I possess in the field of software development, acquired during the formative years at the university, this project came to me as an excellent opportunity for me to achieve a professional and personal goal and integrate a program in a project undertaken by others.

Apart from this, other objectives to be achieved during the preparation of this Final Project are:

- Learn how to use Python's associated libraries such as ReportLab
- Create a real alternative for those users who are not familiar with LaTeX.
- Studying the structure of an alien project and be able to integrate my solution without altering it.
- Learning how to document and argue the completion of a project.

1.3. Memory structure

1. Introduction: This section is intended to give the reader a brief description of the project, explaining the reasons which have impulse it, as well as the goals we hope to achieve during the course of it.

2. State of the art: This chapter will show us the current state of the technologies used during the project and also we will take the opportunity to compare them with other similar technologies. Also we will explain in more detail the project on which we will incorporate our program

3. Requirements: We will define the requirements must meet with our development in order to create a faithful design to the existing one.

4. Design: In this section we will describe how the software implementation has been executed, explaining the decisions taken for the document layout.

5. Evaluation: We will evaluate the goals achieved in regard the objectives that we set at the beginning.

6. Planning and budgeting: At this section we have tried to estimate the time the project should take as well as the total cost of taking it to a commercial environment.

7. Future lines of research: We will recommend the future paths of development that the project could take.

8. Conclusions: Lastly we will analyze the objectives achieved within the complementation of this project, and the possible future lines of development which could be included at the application.

1.4. Glossary of terms

BSD	Berkeley Software Distribution License
CSV	Comma-Separated Values
EPS	Encapsulated PostScript
Eyegrade	Software for MCQ tests
GNU GPL	GNU General Public License
JPEG	Joint Photographic Experts Group standard
LaTeX	PDF layout software
OCR	Optical Character Recognition
ODF	Open Document Format for Office Applications
PIL	Python Image Library
Platypus	Page Layout and Typography Using Scripts
PNG	Portable Networks Graphics
Python	High-level programming language
ReportLab	PDF layout library for Python
RML	Report Markup Language
TTF	True Type Font
UML	Unified Modeling Language
XML	Extensible Markup Language

2. Estado del arte

En este apartado el objetivo es ofrecer una visión global del estado todos los elementos que conciernen al proyecto, tanto en el ámbito de Eyegrade, como en lo que respecta a las tecnologías relacionadas con la maquetación de documentos PDF en Python.

En primer lugar se introducirá al lector a Eyegrade: se contará su historia y el estado en que se encuentra actualmente.

Por otro lado, se presentará cuál es el estado actual de las tecnologías usadas en el proyecto, igualmente se expondrán otras tecnologías similares que se han valorado para el proyecto.

2.1. Eyegrade

2.1.1. Historia

Eyegrade surge como una aplicación con el objetivo primario de proveer una solución de bajo coste para calificar exámenes de opción múltiple, mediante una webcam para puntuar dichos ejercicios. El programa es de software libre y está bajo los términos de la versión 3 o posteriores de la licencia GNU General Public License (GPL). [1]

Existen multitud de aplicaciones que ofrecen un servicio parecido al que plantea Eyegrade, como GEXCAT [2]. Pero la principal diferencia de Eyegrade con respecto a softwares de calificación de este tipo, es la posibilidad de poder utilizar casi cualquier webcam de gama baja, que puedan estar al alcance de todos, en contraste con otras soluciones que se basan en el uso de escáneres más avanzados para el reconocimiento visual de los resultados.

Actualmente el proyecto aún se encuentra en fase alpha, pero es plenamente funcional y, desde el año 2010, ha sido empleado en varias asignaturas de la Universidad Carlos III de Madrid.

Además de la funcionalidad principal descrita, Eyegrade incluye otras características:

- Composición tipográfica de los exámenes: Como ya sabemos, el proyecto provee una solución de generación de los documentos PDF basada en LaTeX, pero es posible generarlos mediante cualquier herramienta. Esto último es lo que

ha dado lugar a la motivación de nuestro proyecto de desarrollar una alternativa a la ya presente.

- Reconocimiento de usuarios: Además de la comprobación de las respuestas marcadas, Eyegrade es capaz de identificar a los alumnos mediante un módulo de reconocimiento de escritura manual de su número de identificación.

- Estadísticas: La aplicación es capaz de ofrecer estadísticas de cada pregunta, con el fin de poder observar que temas son de mayor dificultad para los estudiantes.

- Exportar calificaciones: Por último existe la posibilidad de exportar nuestros resultados a formato *comma-separated values* (CSV), para disponer de los datos y realizar los estudios deseados sobre ellos.

2.1.2. Funcionamiento

Vamos a explicar el funcionamiento del programa en líneas generales para que el lector pueda tener una idea de algunos casos de uso de la aplicación, pero no se entrará en detalles de código ni de jerarquía de clases, ya que eso corresponde al próximo apartado de la memoria (ver capítulo 3).

Vamos a centrarnos en las dos funcionalidades más importantes:

En primer lugar, se tiene la parte de composición del examen, la cual posee un gran interés en nuestro proyecto, puesto que nos ayudará a comprender la funcionalidad básica que tiene nuestra solución a la hora de desarrollarla y el estilo del documento que se va a generar.

En este momento, Eyegrade hace uso de ficheros XML y plantillas de LaTeX para permitir al usuario maquetar el documento a su gusto. En el archivo XML se tendrá que incluir como mínimo la información de las distintas preguntas que constarán de, un texto correspondiente a la pregunta en sí y de las respuestas indicando cuál es la correcta y cuáles no, pudiendo llegar a aportar en el fichero datos como el nombre de la asignatura, el curso al que corresponde dicha asignatura, el título del examen, la fecha del examen, o la duración del mismo. Estos datos no son obligatorios insertarlos en el XML, ya que existe la posibilidad de pasarlos mediante consola al ejecutar el programa.

```
<?xml version="1.0" encoding="UTF-8"?>

<exam xmlns="http://www.it.uc3m.es/jaf/eyegrade/ns/"
      xmlns:eye="http://www.it.uc3m.es/jaf/eyegrade/ns/">

  <subject>Computers and More</subject>
  <degree>An Awesome Degree</degree>
  <title>Final examination</title>
  <date>January 1st, 2011</date>
  <duration>10 min.</duration>

  <question>
    <text>
      What is Python?
    </text>
    <choices>
      <correct>A programming language</correct>
      <incorrect>A computer manufacturer</incorrect>
      <incorrect>A kind of tree</incorrect>
      <incorrect>Who knows!</incorrect>
    </choices>
  </question>

  <question>
    <text>
      What is a webcam?
    </text>
    <choices>
      <correct>A video camera whose output may be viewed in real
         time over a network, especially over the Internet.</correct>
      <incorrect>A video camera intended to spy on spiders</incorrect>
      <incorrect>A kind of fish</incorrect>
      <incorrect>Who knows!</incorrect>
    </choices>
  </question>
```

Ilustración 1: Muestra de archivo XML de entrada

Los exámenes constarán de dos partes principales:

- La portada, en la cual se incluirá toda la información relativa a la prueba (asignatura, curso, fecha, etc.) así como los campos para que el estudiante introduzca sus datos personales, la tabla para la introducción del número de identificación del alumno y la tabla de las respuestas
- Las preguntas, barajadas aleatoriamente según el modelo elegido, que se mostrarán en las sucesivas páginas.

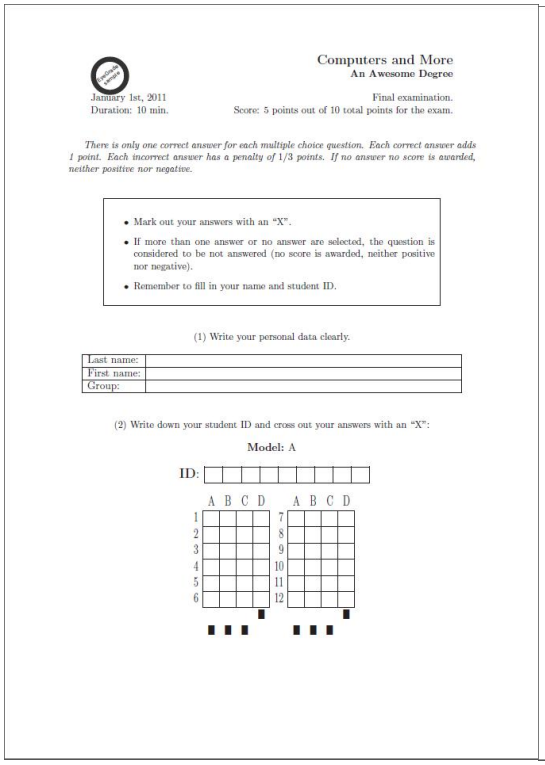
Todo el diseño de la estructura del documento se realizará mediante un archivo de formato LaTeX (.tex), que servirá como plantilla para distribuir los distintos elementos del examen en las páginas del PDF. La más compleja de todas será la portada, ya que tendrá tanto elementos dinámicos que variarán en función de los datos introducidos al programa, como elementos estáticos como avisos, y normas para el examen, que únicamente se podrán modificar en la plantilla LaTeX.

```
\documentclass[a4paper,11pt]{article}

\usepackage{amssymb, amsmath}
\usepackage[utf8]{inputenc}
\usepackage{indentfirst}
\usepackage{multirow}

\usepackage[a4paper, margin=2.5cm, top=2cm, bottom=3cm]{geometry}

\pagestyle{empty}

{{declarations}}

\begin{document}

\begin{center}
\begin{tabular}{p{5.2cm}r}
\multirow{5}{*}[0.35cm]{\scalebox{0.18}{\includegraphics{sample-logo.eps}}} &
\Large  \textbf{{{subject}}} \\
& \textbf{{{degree}}} \\
& \\
{{date}} & {{title}}. \\
Duration: {{duration}}
      & Score: 5 points out of 10 total points for the exam. \\

\end{tabular}
\end{center}

\vspace{0.5cm}

\emph{There is only one correct answer for each multiple choice
  question.  Each correct answer adds 1 point.  Each incorrect answer
  has a penalty of $1/3$ points.  If no answer no score is awarded,
  neither positive nor negative.}

\vspace{0.5cm}

\begin{center}
\begin{tabular}{|p{0.8\textwidth}|}
\hline
\begin{itemize}
\item Mark out your answers with an ``X''.
\item If more than one answer or no answer are selected, the question
  is considered to be not answered (no score is awarded, neither
  positive nor negative).
\item Remember to fill in your name and student ID.
\end{itemize}
\\
\hline
\end{tabular}
\end{center}

\vspace{0.2cm}
```

Ilustración 2: Muestra de plantilla LaTeX

Sin entrar a explicar en detalle las marcas de LaTeX, se puede observar los textos que posteriormente aparecerán en la portada.

Una vez se tienen listos ambos documentos, se procederá a ejecutar el código pasándole como parámetros la plantilla y el fichero XML, entre otros. Un ejemplo de examen generado con el fichero XML mostrado en la Figura 1 y la plantilla de la Figura 2 sería el siguiente:



Computers and More
An Awesome Degree

January 1st, 2011
Duration: 10 min.

Final examination.
Score: 5 points out of 10 total points for the exam.

There is only one correct answer for each multiple choice question. Each correct answer adds 1 point. Each incorrect answer has a penalty of 1/3 points. If no answer no score is awarded, neither positive nor negative.

- Mark out your answers with an "X".
- If more than one answer or no answer are selected, the question is considered to be not answered (no score is awarded, neither positive nor negative).
- Remember to fill in your name and student ID.

(1) Write your personal data clearly.

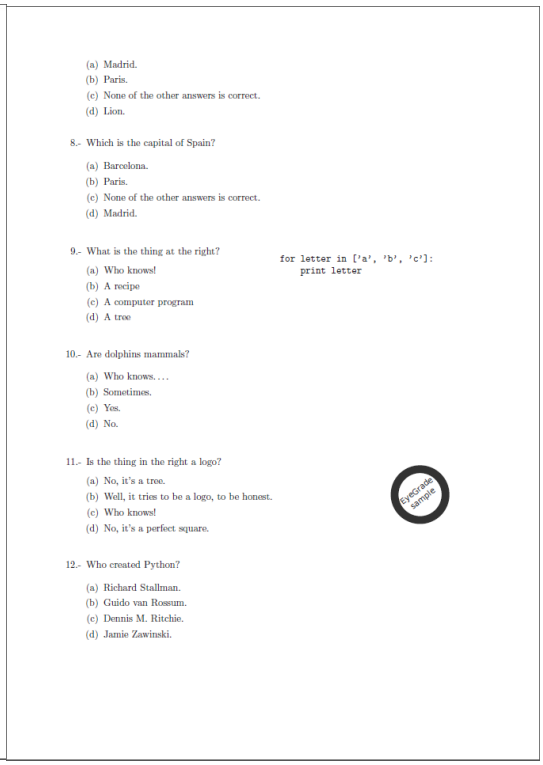
Last name:	
First name:	
Group:	

(2) Write down your student ID and cross out your answers with an "X":

Model: A

ID:

	A	B	C	D		A	B	C	D
1					7				
2					8				
3					9				
4					10				
5					11				
6					12				



(a) Madrid.
(b) Paris.
(c) None of the other answers is correct.
(d) Lion.

8.- Which is the capital of Spain?

(a) Barcelona.
(b) Paris.
(c) None of the other answers is correct.
(d) Madrid.

9.- What is the thing at the right?

```
for letter in ['a', 'b', 'c']:
    print letter
```

(a) Who knows!
(b) A recipe
(c) A computer program
(d) A tree

10.- Are dolphins mammals?

(a) Who knows. . .
(b) Sometimes.
(c) Yes.
(d) No.

11.- Is the thing in the right a logo?

(a) No, it's a tree.
(b) Well, it tries to be a logo, to be honest.
(c) Who knows!
(d) No, it's a perfect square.

12.- Who created Python?

(a) Richard Stallman.
(b) Guido van Rossum.
(c) Dennis M. Ritchie.
(d) Jamie Zawinski.

Fig.3 Fig.4

Ilustración 3: Ejemplo de portada de examen en LaTeX

Ilustración 4: Ejemplo de preguntas en examen de LaTeX

Como se puede observar en esta portada, están presentes los elementos dinámicos aportados por el XML de los que se ha hablado con anterioridad (fecha, duración, asignatura, curso, etc.), y los elementos estáticos provenientes de la plantilla. Los cuadrados negros debajo de la tabla de respuestas, sirven para identificar el modelo del examen.

Se puede apreciar en la Figura 4 superior que, además de las preguntas con sus respectivas opciones, puede haber fragmentos de código o imágenes acompañando a las preguntas.

En cuanto a la corrección de exámenes, al crear un examen se genera un archivo de extensión .eye, que contendrá los datos necesarios para que Eyegrade pueda realizar la calificación. Este fichero contiene unos metadatos del examen tales como: las dimensiones de las tablas, el número de dígitos de la tabla de ID, el modelos de examen, la solución correcta a cada pregunta, y las permutaciones que se han realizado al barajar las preguntas y las opciones. Contando con los datos de ese fichero, y con la base de datos de los alumnos, se procederá a corregir los exámenes enfocándolos con la webcam para la posterior exportación de las calificaciones y sus estadísticas a un fichero CSV.

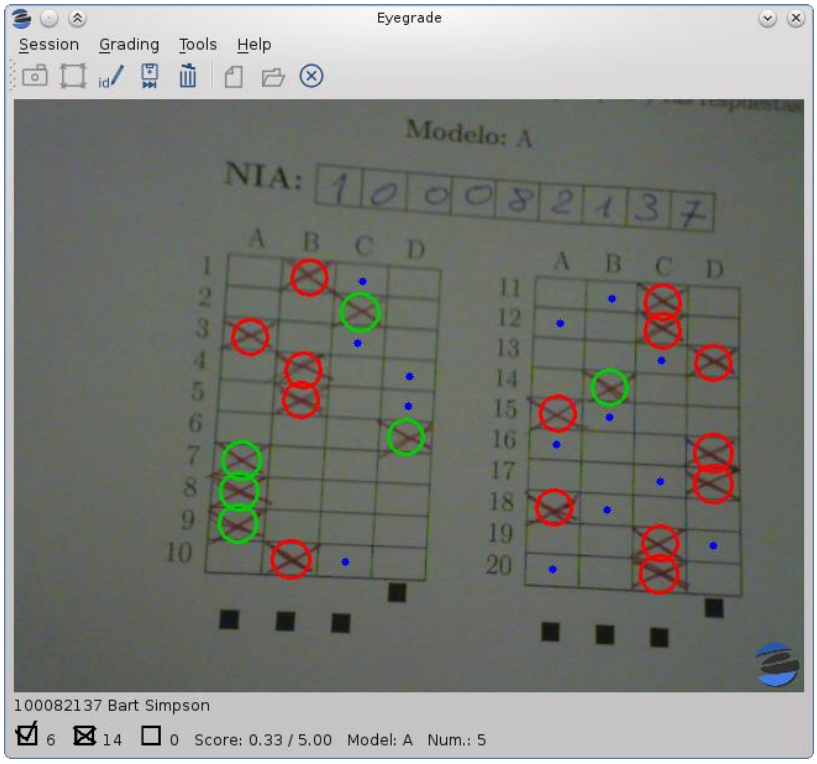


Ilustración 5: Corrección de examen con Eyegrade

El software detectará las opciones escritas, así como el ID del estudiante, entrando de esa forma en el modo revisión de Eyegrade. Para los casos en los que el programa no detecte bien la casilla de ID del alumno, o no detecte tampoco las casillas de respuestas, se disponen de dos modos para solventar estas situaciones:

- Si el ID es lo único que no se ha detectado, una vez dentro del modo de revisión del examen, se permite al usuario seleccionar al estudiante de entre la lista de clase, proponiendo los más parecidos al número reconocido por el *Optical Character Recognition* (OCR).

- En caso de no detectarse ninguna de las tablas, existe un modo manual en la que el usuario habrá de señalar las esquinas de la primera tabla de respuestas y las dos esquinas superiores de la segunda, para ayudar al reconocimiento de las mismas.

Como ya se ha comentado, una vez revisado y corregidos los exámenes, se genera un fichero CSV con las estadísticas de cada alumno en dicha prueba, entre las que se encuentran: ID, modelo de examen, número de respuestas acertadas, número de respuestas erróneas, nota del examen (basada en el peso de las preguntas indicado), y las respuestas dadas a cada pregunta por el examinado.

2.1.3. Análisis interno

Ya se conoce el funcionamiento general de la aplicación gracias a la explicación anteriormente dada, por tanto esta sección servirá para analizar más en detalle la parte del proyecto Eyegrade que afecta a nuestro proyecto.

Es aún más de vital importancia en nuestro caso realizar un buen análisis del comportamiento y, de la estructura y relación del conjunto de clases, puesto que, como se ha comentado anteriormente en este documento, se quiere generar una réplica del programa lo más similar posible a la anterior versión.

Se comenzará analizando cómo está organizada la estructura de la parte del proyecto que nos concierne en lo que a clases y métodos se refiere, de esta forma se podrá ser capaz de comprender como funciona el programa a bajo nivel, y así poder preparar un diseño acorde al existente.

Una vez que sabemos que disposición tiene el código, pasaremos a explicar cómo se ejecuta la aplicación, así como los posibles casos de ejecución y requisitos que nos podremos encontrar según el tipo de datos que le introduzcamos.

2.1.3.1. Estructura

Actualmente Eyegrade cuenta con quince clases que se encargan tanto del proceso de generación de los exámenes, como del reconocimiento de los resultados y su posterior evaluación.

En lo que respecta a nuestro proyecto únicamente necesitaremos analizar dos de ellas: *create_exam*, *exammaker*. Podemos observar su estructura y relación en el siguiente diagrama de clases:

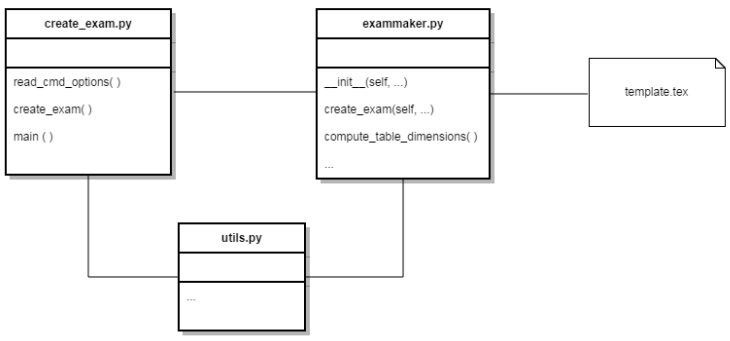


Ilustración 6: Diagrama UML básico de clases antiguo

Pese a que vamos a analizar únicamente dos de ellas, vemos en el diagrama de la figura superior una tercera clase, *utils,* esto se debe a que esta es una clase en la que encontraremos alguna función auxiliar que se llamará desde las dos clases principales, y como vamos a analizar el comportamiento de estas dos, no se ha considerado necesario entrar en el análisis con el mismo detalle como en las otras.

Una vez que tenemos una visión global de la organización de clases, pasaremos a detallar su funcionamiento interno.

- La clase *create_exam* es la clase principal, ese decir, la que se utilizará en la llamada a la hora de ejecutar el código. Esta clase es esencial para comprender el manejo de los datos introducidos por consola, y como se procesan para, realizando las llamadas oportunas, acabar generando el documento.

  - *Read_cmd_options( )*: Como se puede intuir del nombre de esta función, es la encargada de recibir los argumentos que se le pasen por consola durante la ejecución. En primer lugar lo que se hace es, mediante la librería *OptionParser* de Python, añadir todas las posibles opciones que el programa podrá recibir, como pueden ser: el nombre del fichero XML con los datos del examen; o los datos introducidos manualmente; el nombre del fichero de salida; o los distintos modelos que queremos generar de un mismo examen, entre otras opciones.
    Posteriormente se realizan una serie de comprobaciones para garantizar la integridad de la aplicación: que se hayan introducido el mínimo de parámetros esperados; que no se hayan incluido opciones excluyentes.
    En caso de que se produzca algún error, se disponen de mensajes de error para varias de las casuísticas de error valoradas.

  - *Create_exam( )*: Esta función será la encargada de llamar a *read_cmd_options( )*, y de procesar todos los datos recibidos, tanto por el fichero XML, como por argumentos de consola, y, tras crear un objeto con todos los parámetros del examen, llamará y creará un objeto la clase *exammaker*, con el que irá llamando a las funciones necesarias de dicha clase para poder generar el documento.

  - Por último se encuentra el main, que tendrá la misión de comenzar la ejecución del código llamando a *create_exam( )*.

- La clase *exammaker* es la clase en la que se encuentran las funciones sobre las que recaen la misión de maquetar y generar el documento PDF de acuerdo con

los parámetros recibidos a través de las distintas llamadas de la función *create_exam( )* de la clase *create_exam*.
Hay muchos métodos en esta clase que realizan cálculos de dimensiones específicos de LaTeX que no tendrán ningún impacto en nuestro futuro diseño, puesto que se obviará toda la parte relacionada con esa tecnología, y se quedará con el código encargado exclusivamente de la funcionalidad de procesamiento de los datos, y con el encargado de la comprobación de determinadas casuísticas que se pueden dar.
Las funciones más importantes de esta clase son:

- *__init__(self, ...)*: Es la encargada de crear el objeto del examen cuando *create_exam( )* hace una instancia de la clase *exammaker*. Con los datos recibidos, lo que hará será ir procesándolos uno a uno, y se harán varios cálculos como las dimensiones de las tablas, la anchura y altura de las diferentes cuadrículas. La plantilla de LaTeX que se recibe por parámetros se interpreta y divide en base a una variable llamada *re_split_template*, que identificará y separará los distintos elementos de dicha plantilla en base a unos caracteres de escape. Cabe destacar que con un único objeto es suficiente para generar todos los modelos solicitados.
- *Create_exam(self, ...)*: Se realiza una instancia de este método por cada modelo que se vaya generar. La competencias que corresponden a esta función son: crear las distintas tablas en función de los parámetros establecidos previamente haciendo uso de funciones presentes en esta clase, barajar las preguntas dependiendo del modelo, y escribir los datos en el fichero de salida.
- *Compute_table_dimensions( ):*Valora el número de preguntas y opciones de cada pregunta, para calcular las dimensiones de las tablas de respuestas en base a una variable global llamada *param_table_limits* que delimitará el número de filas por cada tabla de respuestas.

Cabe señalar que otra clase importante en el desarrollo es *utils*, pero al tratarse de una clase de apoyo con funciones auxiliares, no se ha considera crítico entrar en detalle de sus distintos métodos, pero es necesario tener conocimiento de su existencia, ya que se realizan llamadas a ella desde las clases arriba descritas y, además, se añadirán las funcionalidades extras necesarias cuando se desarrolle el proyecto.

Con todo lo explicado, una muestra de la ejecución del programa sería la siguiente:

```
C:\Python27\Lib\site-packages\reportlab>python create_exam.py -e exam-questions.
xml -o TEST -m AB template.tex
```

Ilustración 7: Muestra de ejecución del código basado en LaTeX

2.2. Tecnologías y herramientas

2.2.1. Python

2.2.1.1. Historia

Guido van Rossum idea a finales de 1980 Python, y su aplicación se inicia a finales de 1989, por en el CIT en los Países Bajos, como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba. La afición de su creador por los humoristas británicos Monty Python da el nombre al lenguaje.

Fue en 1991, cuando publicó el código de la versión 0.9, y en esta etapa del desarrollo ya estaban presentes clases con herencia, manejo de excepciones, funciones y los tipos modulares, como: strings, listas y diccionarios.

No fue hasta octubre de 2000 cuando se lanza Python 2.0, con nuevas e importantes características, incluyendo un completo recolector de basura y soporte para Unicode. Con este lanzamiento el proceso de desarrollo se cambió, y se hizo más transparente y apoyada por la comunidad de usuarios.

En diciembre de 2008 aparece Python 3.0 (también conocido como "Python 3000" o "Py3k"), una nueva versión del lenguaje que es incompatible con las versiones 2.x. es en su mayoría igual, pero aporta muchos detalles con objetos incorporados como diccionarios y cadenas que cambian considerablemente. Con esta versión se eliminaron una gran cantidad de características en desuso, y la biblioteca estándar se reorganizó. **[3]**

Ilustración 8: Logo Python

2.2.1.2. Características

Python es un lenguaje de programación multiparadigma **[4]**, es decir, permite varios estilos: programación orientada a objetos, programación

imperativa y programación funcional. Además soporta otros paradigmas mediante el uso de extensiones.

A menudo se describe el lenguaje en lo que el desarrollador de Python Tim Peters bautizó como El Zen de Python:

- Hermoso es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Sencillo es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son lo suficientemente especiales para romper las reglas.
- Lo práctico le gana a la pureza.
- Los errores no debe pasar silenciosamente.
- A menos que sean silenciados explícitamente.
- Frente a la ambigüedad, rechazar la tentación de adivinar.
- Debe haber una - y preferiblemente sólo una - manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.
- Ahora es mejor que nunca.
- Aunque "nunca" es a menudo mejor que "ahora mismo".
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede ser una buena idea.
- Los espacios de nombres son una gran idea ¡hay que hacer más de eso!

Python usa tipado dinámico (esto es cuando una misma variable puede tomar valores de distinto tipo en distintos momentos), y cálculo de referencias (técnica para contabilizar las veces que un determinado recurso está siendo referido) para la administración de memoria.

Los tipos de datos existentes se pueden resumir en esta tabla:

Tipo	Clase	Notas	Ejemplo
`str`	Cadena	Inmutable	`'Cadena'`
`unicode`	Cadena	Versión Unicode de `str`	`u'Cadena'`

`list`	Secuencia	Mutable, puede contener objetos de diversos tipos	`[4.0, 'Cadena', True]`
`tuple`	Secuencia	Inmutable, puede contener objetos de diversos tipos	`(4.0, 'Cadena', True)`
`set`	Conjunto	Mutable, sin orden, no contiene duplicados	`set([4.0, 'Cadena', True])`
`frozenset`	Conjunto	Inmutable, sin orden, no contiene duplicados	`frozenset([4.0, 'Cadena', True])`
`dict`	Mapping	Grupo de pares clave:valor	`{'key1': 1.0, 'key2': False}`
`int`	Número entero	Precisión fija, convertido en *long* en caso de overflow.	`42`
`long`	Número entero	Precisión arbitraria	`42L` ó `456966786151987643L`
`float`	Número decimal	Coma flotante de doble precisión	`3.1415927`
`complex`	Número complejo	Parte real y parte imaginaria *j*.	`(4.5 + 3j)`
`bool`	Booleano	Valor booleano verdadero o falso	`True o False`

Tabla 1: Tipos de datos de Python

Python tiene una gran biblioteca estándar, esto es comúnmente citado como una de las mayores fortalezas de Python, estas proporcionan herramientas adecuadas para diversas tareas.

Se puede ver reflejada esa fortaleza en el proyecto, ya que se va a hacer uso de una librería, ReportLab, para desarrollar nuestra aplicación.

2.2.1.3. Versión 2.7 vs 3.3

Como se ha comentado previamente, en diciembre de 2008 se lanzó una nueva versión de Python bajo el nombre clave "Python 3000" o, abreviado, "Py3K". La versión incluye toda una serie de cambios que requieren reescribir el código de versiones anteriores.

Las estadísticas de descargas de Python, en el primer cuatrimestre del 2013, han mostrado que la versión 3.3 finalmente ha sobrepasado las descargas de Python 2.7, por parte de los programadores. [5]

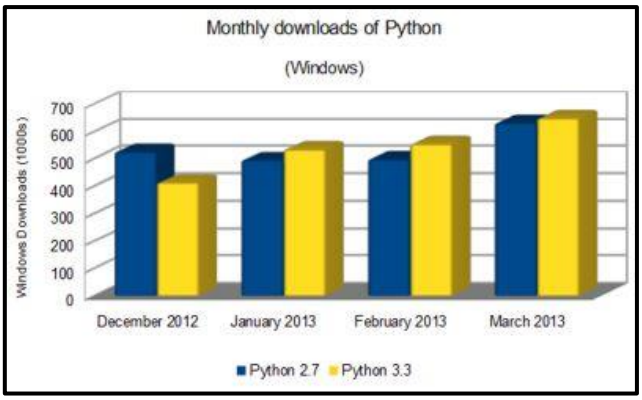


Ilustración 9: Estadísticas de las descargas en versiones de Python

Pese a esto, aún hay quienes no están convencidos y se aferran a la versión 2.7 para trabajar con ella si su desarrollo lo permite, escudándose en la estabilidad de dicha versión como principal argumento.

2.2.2. ReportLab

ReportLab es una biblioteca para generar archivos PDF. Está pensado para solucionar las necesidades de creadores y desarrolladores de webs o diseñadores profesionales que deseen crear o automatizar documentos complejos rápida y fácilmente.

Este generador lleva años creciendo para responder a las necesidades de grandes instituciones financieras y actualmente, no sólo es una biblioteca interesante o un gadget atractivo, sino que se ha convertido en la verdadera solución probada para las empresas.

Un claro ejemplo de éxito es Wikipedia, puesto que cuando se descarga algún artículo en PDF, la herramienta elegida para el renderizado de dicho documento se realiza mediante ReportLab.

La librería pone en práctica un ingenioso y flexible diseño al que llama Platypus, que puede construir documentos a partir de elementos como: titulares, párrafos, fuentes, tablas, gráficos, etc. Esa función asemeja bastante el uso de ReportLab al de LaTeX, ya que simplifica la forma de colocar los elementos, aunque se sigue teniendo un control algo mayor. Además, añade a las 14 fuentes estándar PostScript un soporte completo para Type-1 y fuentes asiáticas.

Además de Platypus, ReportLab pone a disposición de los usuarios el llamado *Report Markup Language* (RML), que basándose en lenguaje de etiquetas similar al de XML, permite maquetar los documentos de forma sencilla. El punto negativo de esta funcionalidad es que se trata de una versión de pago al contrario que el resto de herramientas.

Por otro lado, ReportLab está bajo la licencia Berkeley Software Distribution (BSD), que permite hacer casi cualquier cosa con el código, incluso usarla en un paquete comercial. Como es la parte de desarrollo que se va a realizar en ReportLab la que se va a integrar en un proyecto bajo la licencia GPL, no hay problema de compatibilidad. El problema hubiera surgido si se hubiera tratado de introducir código GPL en un programa bajo licencia BSD.

Por último, cabe resaltar que ReportLab genera directamente los documentos en PDF, mientras que LaTeX crea un archivo que necesitará ser convertido a PDF.

Ilustración 10: Logo ReportLab

2.2.3. Tecnologías alternativas

2.2.3.1. LaTeX

LaTeX es un sistema de composición de textos, orientado especialmente a la creación de libros, documentos científicos y técnicos que contengan fórmulas matemáticas. Con código abierto y basado en instrucciones que se centra en los contenidos. **[6]**

Sus ventajas más destacadas son su estabilidad y multiplataforma, la alta calidad en la edición de ecuaciones, facilidad en la construcción de macros y órdenes, se escribe en ASCII y es gratuito.

Como inconvenientes nos encontramos su complejidad en el uso, poca variedad de tipos de fuentes, no permite directamente crear archivos PDF, la limitación en la edición al utilizar sus macros, y la necesidad de realizar la instalación.

2.2.3.2. DocBook

DocBook crea documentos en un formato neutro, independientemente de la presentación, donde se recogen tanto el contenido como la estructura lógica del mismo. **[7]**

Ventajas a destacar: es gratuito, permite que el documento pueda ser publicado automáticamente en multitud de formatos sin que sea necesario ningún cambio sobre el documento original. El código tiene un aspecto similar a HTML pero su formato XML ofrece muchos más.

Entre sus inconvenientes se encuentran la gran rigidez de la estructura que dificulta la escritura, carece de soporte nativo para bibliografías aunque existen extensiones más o menos acertadas, y es difícil integrar contenidos, como los listados con formato.

2.2.3.3. ODF

Open Document Format for Office Applications (ODF) es un formato de archivo abierto y estándar para el almacenamiento de documentos ofimáticos tales como hojas de cálculo, textos, gráficas y presentaciones. Open Office tiene la misma funcionalidad que Microsoft Office, con una gran diferencia en el precio. **[8]**

Cuenta con ventajas como su gratuidad en las actualizaciones, es compatible con MS Office (todas las versiones), puede exportar documentos a PDF con un sólo click sin necesidad de instalar nada más, es muy estable y usa menos espacio en disco duro.

Resaltar como inconvenientes y críticas que tiene algunos problemas para mostrar correctamente fórmulas matemáticas, y que la especificación ISO OpenDocument no permite tablas en presentaciones.

2.2.3.4. Conclusiones

Tras analizar estas alternativas, con sus respectivas ventajas e inconvenientes, se ha decidido optar por ReportLab ya que puntos como su integración con Python, sus posibilidades de maquetación a alto y bajo nivel de los documentos, la posibilidad de crear directamente el documento PDF, el respaldo de la comunidad de desarrolladores y la confianza depositada por empresas como Wikipedia, hacen que la herramienta se adapte bien a las necesidades del proyecto.

3. Requisitos de la aplicación

En esta sección identificaremos los requisitos que tendrá que tener en cuenta nuestro diseño para recrear el funcionamiento de la solución basada en LaTeX, así como para tratar de asemejar lo máximo posible la lógica de ejecución ya establecida:

- Será necesario preservar la estructura de los archivos XML de entrada, y en la medida de lo posible, los datos introducidos en él.

- Habrá que buscar e incluir la misma tipografía que usa LaTeX, de esta forma mantendremos un estilo uniforme y no habrá una diferencia significativa entre las dos soluciones.

- Las fuentes a usar para los bloques de código ha de ser una tipografía de ancho fijo.

- Será imprescindible la posibilidad de incluir imágenes, a poder ser en forma de gráficos vectoriales (como formatos EPS, PDF, etc.).

- Los modelos de examen que se podrán generar estarán comprendidos entre los valores A-H, correspondiéndoles a cada modelo un distinto mezclado del orden de las preguntas y sus opciones. Tendrá que existir también la posibilidad de generar un modelo 0, el cual mantendrá el orden de las preguntas exactamente como se han provisto en el fichero XML, teniendo que hacer lo mismo con las respuestas, con la excepción de que la respuesta correcta sea siempre la primera opción.

- Cada modelo tiene que llevar asociado cuatro cuadrados negros que se colocará debajo de las tablas de respuestas, y servirán para identificar el modelo de examen.
  Los tres primero valdrán para reconocer el modelo, y el cuarto será redundante respecto a los otros tres. Los cuadrados podrán tener dos posiciones: arriba y abajo; y puesto que hay tres cuadrados para identificar el modelo con dos posibles posiciones, da lugar a ocho posibles combinaciones, de ahí que actualmente sean soportados únicamente modelos de la A a la H (las ocho primeras letras).
  El reparto de los patrones de los cuadrados vendrá impuesto por la siguiente tabla:

Model				
A	Down	Down	Down	Up
B	Up	Down	Down	Down
C	Down	Up	Down	Down
D	Up	Up	Down	Up
E	Down	Down	Up	Down
F	Up	Down	Up	Up
G	Down	Up	Up	Up
H	Up	Up	Up	Down

Ilustración 11: Modelo de ordenación de los cuadrados negros según su modelo

- A la hora de generar la parte de las preguntas, será obligatorio cubrir todos los posibles casos en los que se pueden presentar los textos de las preguntas y los de las opciones:

  - Los textos de las preguntas podrán ir acompañados de: un bloque de código, cuya identación deberá ser respetada para conservar su esencia; o bien de una imagen.
  - Las opciones podrán estar formadas por: texto, código o imágenes; pero nunca más de un tipo a la vez.

- Los textos tanto de las preguntas, como de las respuestas actualmente pueden tener marcas LaTeX para realizar modificaciones en los estilos de las fuentes. Habrá que valorar la posibilidad de desarrollar dicha funcionalidad.

4. Diseño

4.1. Consideraciones previas

Antes de empezar a diseñar la solución y, conociendo tanto las posibilidades que ofrece la librería ReportLab como las metas necesarias de alcanzar propuestas en el apartado de requisitos, se ha empleado tiempo en realizar un estudio y valoraciones antes de comenzar a desarrollar propiamente la aplicación.

Esta planificación nos ayudará a optimizar el tiempo, ya que evitaremos perder tiempo en el futuro cuando nos encontrásemos esos inconvenientes. En su lugar emplearemos ese tiempo desarrollando una solución ya contemplada.

4.1.1. Limitaciones

Sabemos por lo explicado en anteriores capítulos, los documentos generados con la versión de LaTeX hacían uso de imágenes que podrán usarse como logo para la portada, como gráfico acompañando al texto de una pregunta, o bien usar dichas figuras como opciones disponibles para una pregunta.

Como se ha especificado en los requisitos, en la anterior versión se hacía uso de gráficos vectoriales para representar esas imágenes, en concreto el formato EPS, que son ampliamente soportados por LaTeX.

Aquí nos enfrentamos a la mayor limitación que nos ofrece la alternativa de ReportLab frente a la ya existente, que no es otra que la no posibilidad de hacer uso de gráficos vectoriales en sus documentos. ReportLab de forma nativa solamente admite el formato JPEG.

La solución que se ha planteado para este problema ha sido sustituir los gráficos vectoriales por imágenes en formato PNG, que serán soportados gracias a la inclusión de la librería *Python Imaging Library* (PIL), encargada del manejo de imágenes en Python.

Pese a que la calidad se verá algo resentida, no supondrá un gran impacto en el resultado final, y habremos salvado el primer gran problema.

4.1.2. Diseño conceptual

Se tratará de describir en esta sección las ideas del diseño que se pudieron planificar antes de comenzar a desarrollar una vez estudiada la estructura de Eyegrade.

4.1.2.1. Idea general

En este apartado se quiso establecer una serie de requisitos, que sirvieran como guía de lo que se quería hacer con el código ya existente. Las ideas planteadas fueron:

- Reutilizar la mayor parte de código posible: puesto que lo importante en este proyecto es la maquetación del documento, se tratará de mantener, en la medida de lo posible: el código de arranque desde consola, el parseado de los argumentos y su subsiguiente uso, la generación del objeto de examen y su posterior manejo para usarlo en la llamada en la generación del documento.

- Usar la herramienta Platypus de ReportLab: Platypus es una librería de alto nivel de maquetación de ReportLab que permite crear documentos complejos con poco esfuerzo. Se ha empleado debido a la simplificación que plantea a la hora de modificar estilos de los distintos elementos, ya que pese a que ReportLab nos permite manejar y ubicar los distintos componentes de nuestro trabajo, para un trabajo como el nuestro en el que repetiremos acciones de cambio de fuente, creación de tablas, etc., Platypus es mucho mejor opción.

- Realizar una plantilla única: pese a que puede no parecer la mejor opción por el desorden que se puede producir de código fuente en la clase, se ha decidido aunar todo el diseño de la maquetación en una sola, este diseño será mejorable en el futuro, pero permite a un usuario ajeno a localizar todos los elementos usados en la misma clase de una forma sencilla.

4.1.2.2. Fuentes

Como se había indicado en los requisitos, las fuentes usadas en nuestro desarrollo tendrán que ser las mismas que en LaTeX.

Tras un estudio en la red, se descubrió que la fuente nativa de dicha herramienta era la familia Latin Modern Roman, que estaba disponible para su descarga gratuita, por lo que se procedió a almacenar los archivos TTF, que son los que necesita ReportLab para añadir una fuente externa.

En cuanto al otro requisito relacionado con las fuentes, se había requerido que para los bloques de código se empleara una fuente de ancho fijo. Se ha elegido la fuente Courier,

ya que ReportLab la incluye de forma nativa, y es la fuente para código por antonomasia.

4.1.2.3. Wrapper

Hemos comentado que vamos a utilizar Platypus para simplificar el proceso de maquetación, pero, buscando por Internet información y manuales de esta herramienta, se ha encontrado una librería apodada como PDF, elaborada por el usuario H.C. v. Stockhausen y abierta al uso para cualquier persona, que ayuda a simplificar aún más el proceso de cambiar los estilos y manejar las imágenes, párrafos y demás elementos. **[9]**

Por este motivo se ha decidido incluir y emplear en el desarrollo.

4.2. Implementación

En este apartado finalmente se explicará las decisiones, el desarrollo, y las modificaciones realizadas en el código.

Se ha decidido desglosar las decisiones de diseño en función de cada clase, explicando las modificaciones realizadas en el caso de que ya existiese dicha clase (el caso de todas menos de *template*), y/o el nuevo desarrollo llevado a cabo. Asimismo se incorporarán imágenes y diagramas explicativos.

4.2.1. Create_exam.py

Partiendo de la base de la idea introducida por el punto 4.1.2.1., de la presente memoria, el objetivo en esta clase es hacer uso de la mayor parte de la funcionalidad posible.

Como ya se ha explicado, en esta clase se realiza la lectura de los argumentos introducidos en consola durante la ejecución del programa, y tras su posterior manipulación, se crea una instancia de la clase *exammaker*, que se encargará de enviar los datos al maquetador.

El punto que se ha tenido en cuenta a la hora de modificar esta clase ha sido únicamente el suprimir lo relacionado con LaTeX.

En este caso se ha eliminado en la función *read_cmd_options( )* la petición del argumento *template* así como la comprobación de que se hubiera introducido dicho

parámetro, ya que a partir de ahora la plantilla que se usará estará descrita en el código de la clase con el mismo nombre.

Asimismo, en la llamada realizada en el método *create_exam( )* para crear el objeto de la clase *exammaker*, se ha quitado también el argumento *template*, siendo innecesario a partir de ahora por la misma razón que antes.

La estructura de la clase queda intacta, pese a que haya habido unas ligeras modificaciones en su código para adaptarlo a las nuevas necesidades.

Una muestra de ejecución de la aplicación con el nuevo código quedaría de la siguiente manera:

Ilustración 12: Muestra de ejecución con el nuevo código

4.2.2. Exammaker.py

Con esta clase se aplicará un criterio similar al usado con *create_exam*, a diferencia de que en esta se hallan multitud de funciones relativas a cálculos de dimensiones específicas para LaTeX, por lo que el código se verá drásticamente recortado hasta dejar la funcionalidad necesaria.

- En cuanto a variables globales se han modificado los valores de *param_table_limits* reduciendo los valores de la lista en 1, ya que había una error de cálculo a la hora de limitar el número de filas por tabla, debido a que en el bucle en el que se comprueba dicha variable se utiliza el comparador >=, y se producía un desborde. Se ha eliminado la variable *re_split_template* ya que no hará falta interpretar ninguna plantilla de LaTeX.

- En la función inicializador *__init__( )* no se ha modificado el código más allá de eliminar de los argumentos esperados la variable *template* para que resulte congruente con lo hecho en el apartado en la llamada a dicho método en la clase *create_exam( )*.

- En *create_exam( )*,se siguen procesando todos los datos que le vamos a pasar al maquetador de exámenes, no obstante se han cambiado la llamada a dicho generador, ya que ahora esa responsabilidad recae sobre el método *generate_pdf( )*, correspondiente a la clase *template*.
  En esa llamada irán incluidos los bits resultantes de la función *encode_model*, correspondiente a *utils*, estos bits lo que hacen es dar la información necesaria

para generar los cuadros negros de debajo de las tablas de respuestas, en función del modelo que se decida codificar.
Por último resaltar que para escribir toda la información en el documento se hace uso de una nueva función llamada *write_binary_files( )*, perteneciente a la clase *utils*, y que explicaremos el porqué de esta función y su misión.
Al igual que en anteriores clases, se ha eliminado todo el código innecesario relacionado con LaTeX. En esta clase es especialmente llamativo ya que únicamente se mantendrán 9 métodos de los 22 existentes, entre los que se salvan se encuentran los que realizan cálculos de que las dimensiones de la tabla sean correctas.

4.2.3. Template.py

Este apartado es esencial para entender todo el desarrollo, en él describiremos en detalle cómo se ha ido resolviendo cada parte del documento, explicando las partes que más problemas plantearon, y comentando como se solventaron dichos obstáculos.

Para hacer más legible el texto, se ha decidido desglosar las explicaciones en tres bloques: Uno general dónde se explicarán decisiones de diseño globales, y por otro lado los dos grandes partes del documento: la portada y las preguntas.

Todo el diseño del examen recae sobre la función *generate_pdf( )*, que recibirá como argumentos las preguntas del examen, el modelo, un booleano indicando si el examen a generar es con solución o no, y los bits para generar los cuadrados de negros de identificación del modelo.

4.2.3.1. General

Lo primero que se hace en el código tras realizar las importaciones oportunas, es importar las fuentes al proyecto. Este proceso se consigue gracias a la función *registerFont( )* importada de una sublibrería de ReportLab. Para la representación de los distintos estilos de la fuente Latin Modern Roman utilizados en el documento ha sido necesario descargarse los siguientes paquetes de fuentes TTF: Latin Modern Roman, para los textos normales; Latin Modern Roman Dunhill, una fuente más estrecha que la normal, que se usará en la nomenclatura de las filas y columnas de la tabla de respuestas; Latin Modern Roman Bold, para los textos en negrita; Latin Modern Roman Italic, para los textos en cursiva.

Una vez se han importado las fuentes, se procede a crear una clase de estilos, a la que se ha llamado *MyTheme*, que incluya todos los posibles estilos de párrafo que nos vamos a

ir encontrando a lo largo del documento. Estos estilos se fueron creando, probando y depurando a lo largo de las pruebas realizadas durante la creación de la plantilla.

Al final del desarrollo se han generado 16 estilos distintos que cubren las necesidades concretas de partes del documento como son: Títulos, subtítulos, cursivas, negrita, código fuente (usando Courier como ya se había adelantado al comienzo del capítulo de diseño), etc.

La sencillez de Platypus permite que, una vez se le ha otorgado un nombre a un estilo concreto, basta con pasarlo por parámetro cuando se esté añadiendo texto al documento, para aplicarlo.

En segundo lugar, se ha decidido crear la variable global t*emplateData* que será un diccionario de Python que incluya todos los textos que se van a usar en la portada, tanto los estáticos como los textos que recibirán de forma dinámica datos como la duración del examen, el nombre de la asignatura, etc. También se ha añadido en el diccionario el nombre del fichero de imagen del logo.

Esta decisión hace muy fácil la posible edición, adición o eliminación de textos sin tener que navegar por el código.

4.2.3.2. Portada

La portada ha supuesto una parte compleja debido a la cantidad de contenido que en ella se muestra, mezclando a su vez variables con contenido estático y dinámico, además de las múltiples tablas.

Se analizará por bloques poniendo como ejemplos capturas de pantalla del resultado final de cada conjunto y explicando las decisiones de diseño que se han llevado a cabo con cada una.

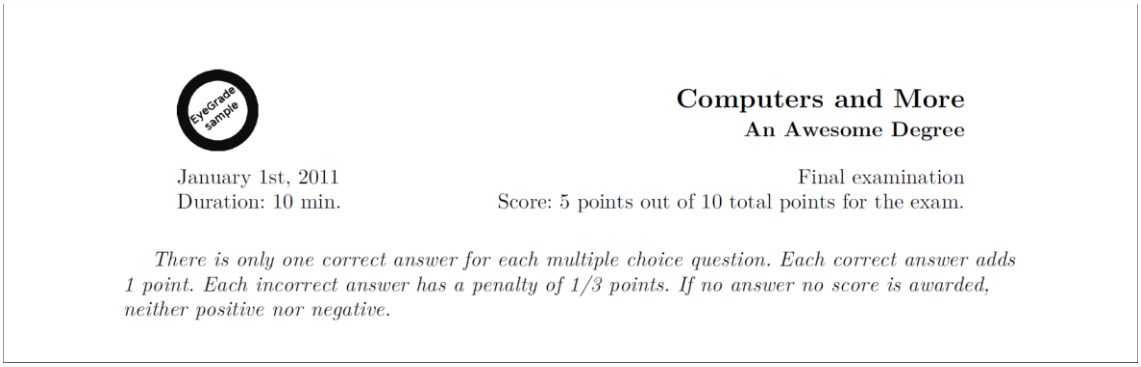


Ilustración 13: Bloque 1 de la portada generada

Como se puede observar en este primer bloque, ya nos encontramos con multitud de elementos distintos.

A lo largo del diseño se ha tratado de agrupar los elementos en tablas por su facilidad de posicionamiento. Gracias a Platypus y su posibilidad de producir estilos diferentes, para cada tabla, se ha podido acomodar los elementos en función de las necesidades que tuvieran.

Así pues, la primera tabla presente en la portada incluye todos los elementos visibles en la figura superior, a excepción del texto de descripción que aparece en cursiva. Tras obtener los datos de *templateData* y preparar los distintos elementos, como el logo, fecha, duración, título de la asignatura, subtítulo, nombre del examen e información, y aplicarles los respectivos estilos de párrafo, que se han mencionado en el apartado anterior, se coloca cada elemento en una posición de la tabla y, tras ajustar las medidas y posiciones, se añade al documento.

En segundo lugar tenemos una descripción del peso de las preguntas en el examen escrito en cursiva. Simplemente se ha añadido una separación vertical respecto a la tabla anterior, y tras extraer el contenido del diccionario relativo a este texto, se añade en un párrafo con un estilo generado que incluye la fuente cursiva y centrada.

- Mark out your answers with an “X”.
- If more than one answer or no answer are selected, the question is considered to be not answered (no score is awarded, neither positive nor negative).
- Remember to fill in your name and student ID.

Ilustración 14: Bloque 2 de portada generada

En la figura del segundo bloque observamos dos nuevos elementos: los bordes del recuadro, que se establece en la configuración de la tabla en la que se han incluido los textos, y la inclusión de puntos de lista, que se han definido en un estilo de párrafo.

(1) Write your personal data clearly.

Last name:	
First name:	
Group:	

(2) Write down your student ID and cross out your answers with an “X”:

Ilustración 15: Bloque 3 de portada generada

En este bloque intermedio podemos observar dos párrafos que seguirán la misma lógica aplicada a los anteriores. La tabla 3x2 que se encuentra en medio, se ha realizado aplicándole estilo a dicha tabla, para rellenar los bordes tanto externos como internos.

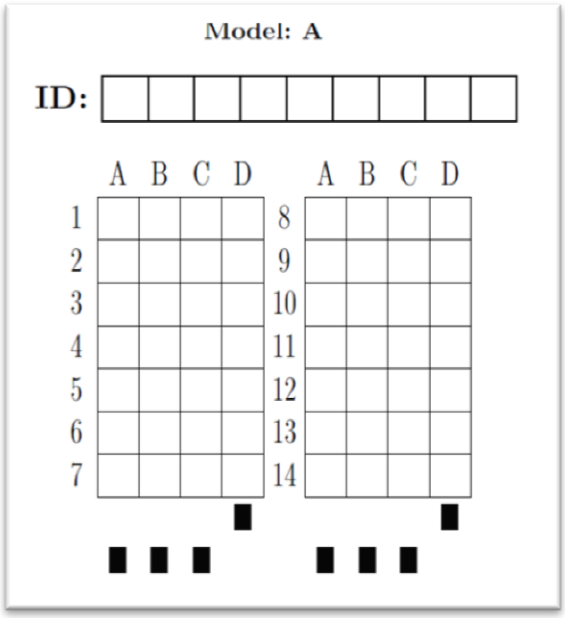


Ilustración 16: Bloque 4 de portada generada

El último bloque de la portada es con diferencia el más complejo, ya que la generación de las tablas dependerá de varios algoritmos que tendrán en cuenta todos los cálculos de dimensiones realizados en la clase *exammaker*, teniendo que respetar las normas de ancho que establece Eyegrade para el conjunto de este bloque.

La generación de los cuadrados negros para la identificación del modelo, se han realizado utilizando el la variable de *bits* que se la pasa por parámetro a la función *generate_pdf( )*. Usando un algoritmo para calcular si el cuadrado corresponde arriba o abajo, y aplicándole un desplazamiento en cuestión.

4.2.3.3. Preguntas

Del mismo modo que se ha explicado los bloques de la portada, se pasará a comentar los posibles tipos de pregunta que se podrán dar en un examen.

4.- What is the thing at the right?

```
for letter in ['a', 'b', 'c']:
    print letter
```

(a) A tree
(b) A computer program
(c) A recipe
(d) Who knows!

Ilustración 17: Pregunta con código de apoyo al enunciado

6.- Is the thing in the right a logo?

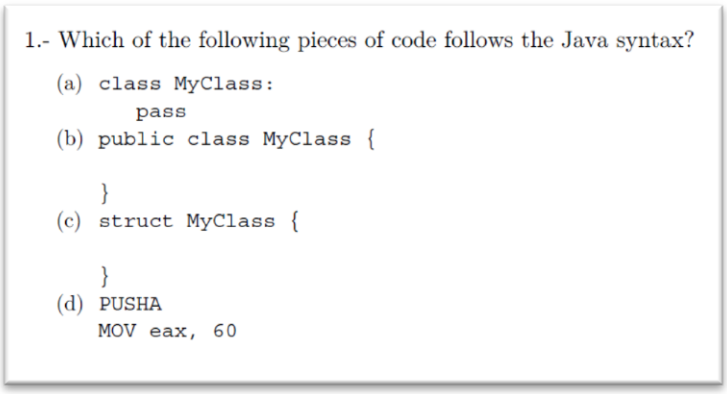
(a) No, it's a tree.
(b) Who knows!
(c) Well, it tries to be a logo, to be honest.
(d) No, it's a perfect square.

Ilustración 18: Pregunta con imagen de apoyo al enunciado

Para estos dos tipos de preguntas, la solución ha sido muy parecida, ya que se ha agregado a la derecha del enunciado el elemento detectado.

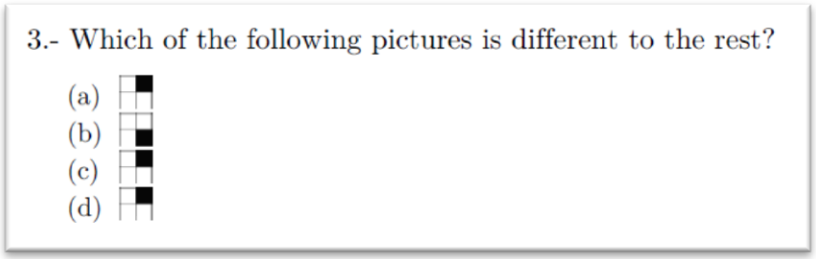
La complejidad en ambos casos ha sido la de encontrar una solución para su posicionamiento. Al final se decidió encapsularlos en una tabla, para mayor facilidad de su manejo en de la posición.

Uno de los problemas más complejos que han surgido ha sido la necesidad de mantener la identación de los bloques de código, bien como apoyo al enunciado, o bien como opción de las preguntas. Tras un estudio en profundidad de los datos que se reciben cuando se usa la etiqueta *<code>* en el fichero XML, se ha llegó a una solución de buscar los espacios en blanco en el texto del código, y reemplazarlos por *non breaking spaces ( ),* y de los retornos de carro (\n) por *line breaks (<br>)* para asegurar la integridad del código fuente empleado. Se encontraron problemas con algunos bloques de código con la identación de la primera línea correctamente, y se diseñó una solución para salvar este escollo y preservar de esta forma la tabulación.


1.- Which of the following pieces of code follows the Java syntax?

(a)
```
class MyClass:
    pass
```
(b)
```
public class MyClass {

}
```
(c)
```
struct MyClass {

}
```
(d)
```
PUSHA
MOV eax, 60
```

Ilustración 19: Pregunta con código en las opciones


3.- Which of the following pictures is different to the rest?

(a)
(b)
(c)
(d)

Ilustración 20: Pregunta con imágenes en las opciones

Por último, los casos más extraños que se podían dar en las preguntas era que las opciones incluyesen código o imágenes.

Con el código la complejidad se reducía al mismo asunto arriba descrito para usar bloques de código: respetar la identación del código, para lo cual se utilizó la misma solución que se había diseñado para los bloques de código cuando aparecían acompañando a enunciados.

En este caso causaron más problemas las imágenes, por lo que se decidió introducirlas individualmente en una tabla para facilitar su colocación y manejo del tamaño.

Tras haber procesado todas las preguntas del fichero, se devuelve un objeto con el documento renderizado.

La estructura final quedaría de la siguiente manera:

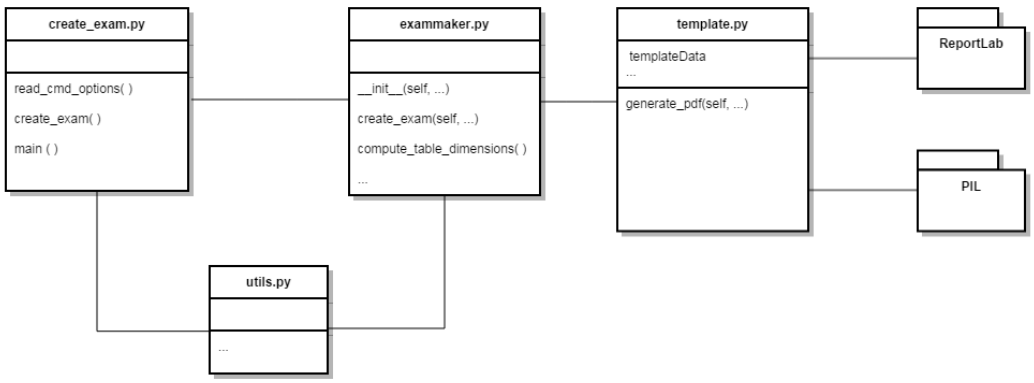


Ilustración 21: Diagrama UML de clases nuevo

4.2.4. Utils.py

Como se ha visto anteriormente, la clase *utils* almacena funciones auxiliares. Para solventar un problema de visualización de los documentos PDF cuando se iba a escribir los datos en él, se ha creado la función *write_binary_files* ya que como los datos que se van a escribir ahora son binarios la función que antes escribía los datos en LaTeX no era válida.

5. Evaluación y resultados

Durante el desarrollo de este trabajo fin de grado, se estableció el criterio de evaluar el funcionamiento del programa de manera progresiva. Bajo esta premisa se pudo ir comprobando que las funciones que se implementaban se comportaban de la manera esperada.

En el momento en el que el programa se encontraba en su estadio final, todas las funciones que se habían fijado como requisitos habían sido implementadas a excepción de la posibilidad de usar marcas del lenguaje, que por decisiones de tiempo y complejidad se decidió establecerlo como una posible línea de trabajo futuro.

Para comprobar la robustez del sistema, se han realizado múltiples pruebas con distintas casuísticas, para comprobar que el programa funcionaba en distintos entornos de ejecución.

Para aportar un punto de vista ajeno al desarrollador, se probó el programa con posibles usuarios finales. De esta manera se aporta verosimilitud a las pruebas y, gracias a sus experiencias de uso, subsanar cualquier error obviado.

6. Planificación y presupuesto

6.1. Planificación

Para la consecución de este proyecto se han llevado a cabo varias fases dependientes unas de otras, las cuales serán explicadas a continuación de manera cronológica:

Fase 1: Búsqueda de una alternativa

Como objetivo principal de este proyecto, se propuso realizar una alternativa real para la maquetación de exámenes tipo test para el proyecto Eyegrade para ofrecer una alternativa a aquellos usuarios que no estuviesen familiarizados con LaTeX. Para ello, se tuvo que plantear una solución que fuera plausible para su desarrollo, que estuviese dentro del marco del software libre y que pudiese servir como punto de partida para que cualquier desarrollador, conocedor de la tecnología, sea capaz de mejorar el programa ante cualquier necesidad futura. Mediante un estudio de mercado comprobamos las posibilidades que existen como alternativa a este programa y se analizó en este contexto el papel que Eyegrade desempeña. A partir de todas estas indagaciones fue posible establecer unos objetivos que sirven de base a este proyecto.

Fase 2: Estudio tecnologías actuales

Tras establecer la idea de crear una alternativa a una herramienta basada en LaTeX, se buscó la tecnología que fuese más adecuada para nuestro propósito. En este punto se presentó ante nosotros la posibilidad de usar ReportLab, dentro del proyecto Eyegrade. Gracias a la investigación llevada a cabo sobre Python y en especial de ReportLab, se consiguieron los recursos necesarios para entrar con profundidad al aprendizaje de estas.

Fase 3: Aprendizaje Python y ReportLab

Esta fase se presenta como esencial para el alcance de nuestro objetivo. Python es un lenguaje desconocido para el desarrollador por lo que durante este tiempo se trabajó de manera incesante en el aprendizaje de este, haciendo hincapié en la librería que resulta de mayor importancia en el proyecto, ReportLab. Gracias al trabajo de investigación con documentación experta acerca de este lenguaje de programación y de Eyegrade, junto con las continuas pruebas para afianzar conocimientos, se consiguió llegar a un punto de competencia apropiado para la realización del programa.

Fase 4: Diseño conceptual del programa

Como paso previo al desarrollo del programa, se precisó un tiempo para establecer los elementos que debe constar el mismo. Se planteó un diseño robusto,

fácilmente integrable con Eyegrade y con una estructura del sistema que fuese fácilmente interpretada por aquellos futuros desarrolladores que quieran extender o mejorar su funcionalidad. Se tuvo en cuenta al público al cual va dirigido con el fin de establecer un modelo adecuado. Esta fase nos ayuda a afianzar las ideas sobre nuestro programa y establece un punto de partida óptimo para comenzar la fase de desarrollo.

Fase 5: Desarrollo del programa

Tras el imperioso trabajo de aprendizaje de las tecnologías que se van a usar y el posterior diseño conceptual del programa, comienza la fase de desarrollo. Como sección vertebral del desarrollo, la creación de la clase *template* supuso la rama principal sobre la cual se fueron integrando las clases esenciales de este programa: *create_exam* y exammmaker, junto con las clases auxiliares como *utils.* Durante esta etapa fue necesario el uso constante de la documentación, al igual que se establecieron continuas pruebas que demostrasen el correcto funcionamiento del programa.

Fase 6: Fase de pruebas

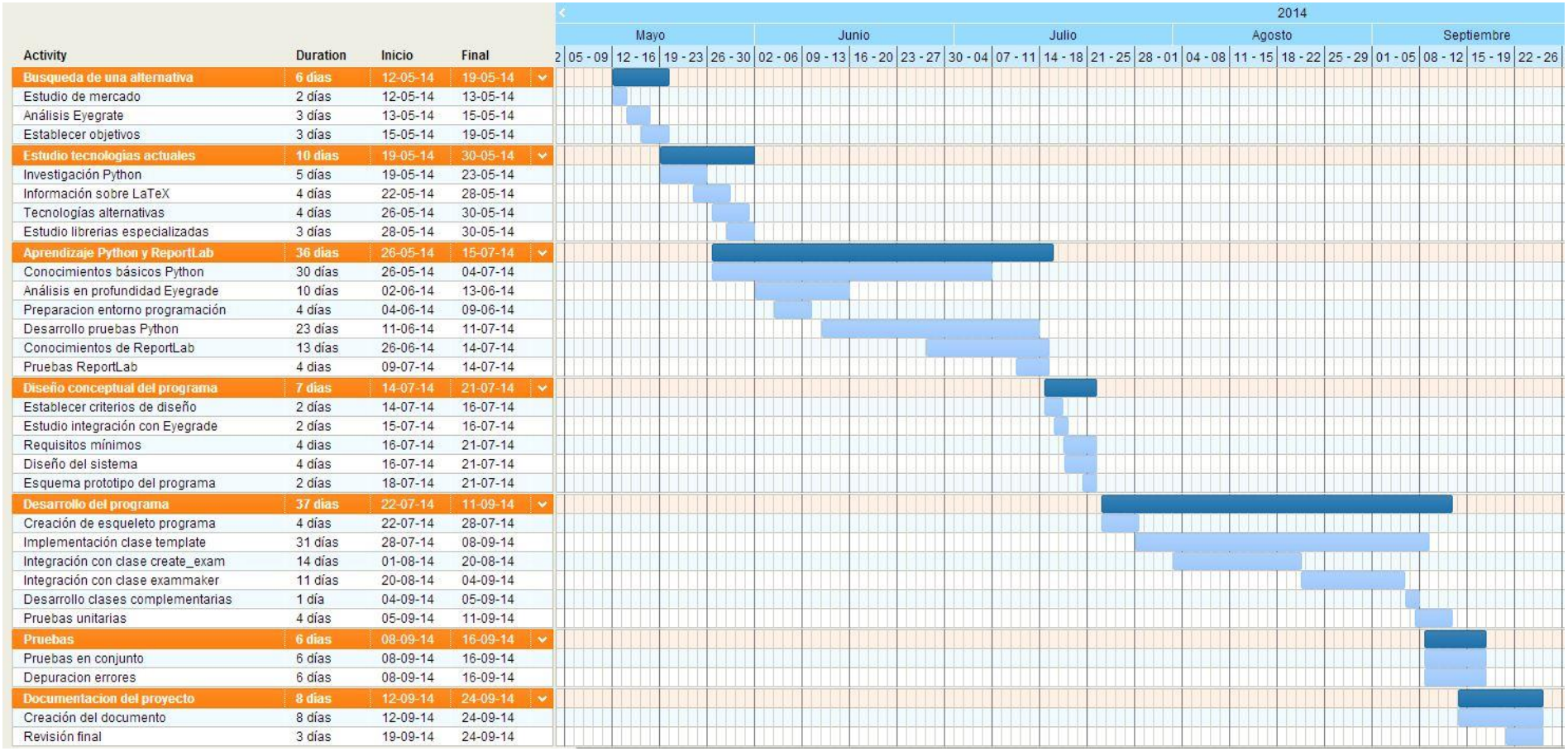
Con la finalización del grueso del programa fue necesario establecer un periodo de pruebas. En él se contemplaron aquellos casos que durante la fase de desarrollo no se tuvieron en cuenta y se fueron depurando los errores que se observaron durante este tiempo. Al finalizar esta fase tuvimos una versión del programa sólida y depurada, lista para ser usada.

Fase 7: Documentación del proyecto

Como fase final de este proyecto se reunió información de todas las etapas de las cuales consta el mismo y se ha realizado la presente memoria, con el fin de dar a conocer toda la indagación y trabajo que se ha llevado a cabo para ejecutar el proyecto. Durante la presente fase fue necesaria la intervención constante del tutor, quien aportó las bases para la presentación de esta tarea y la rectificación de aquello que no aportaba información esencial del programa o de información que nutría al lector de una visión más completa de este Trabajo de Fin de Grado.

Diagrama de Gantt

Para representar de manera más concreta la repartición de las fases y el tiempo necesario para cada una de ellas se presenta a continuación un diagrama de Gantt:



Activity	Duration	Inicio	Final
Busqueda de una alternativa	**6 días**	**12-05-14**	**19-05-14**
Estudio de mercado	2 días	12-05-14	13-05-14
Análisis Eyegrate	3 días	13-05-14	15-05-14
Establecer objetivos	3 días	15-05-14	19-05-14
Estudio tecnologías actuales	**10 días**	**19-05-14**	**30-05-14**
Investigación Python	5 días	19-05-14	23-05-14
Información sobre LaTeX	4 días	22-05-14	28-05-14
Tecnologías alternativas	4 días	26-05-14	30-05-14
Estudio librerias especializadas	3 días	28-05-14	30-05-14
Aprendizaje Python y ReportLab	**36 días**	**26-05-14**	**15-07-14**
Conocimientos básicos Python	30 días	26-05-14	04-07-14
Análisis en profundidad Eyegrade	10 días	02-06-14	13-06-14
Preparacion entorno programación	4 días	04-06-14	09-06-14
Desarrollo pruebas Python	23 días	11-06-14	11-07-14
Conocimientos de ReportLab	13 días	26-06-14	14-07-14
Pruebas ReportLab	4 días	09-07-14	14-07-14
Diseño conceptual del programa	**7 días**	**14-07-14**	**21-07-14**
Establecer criterios de diseño	2 días	14-07-14	16-07-14
Estudio integración con Eyegrade	2 días	15-07-14	16-07-14
Requisitos mínimos	4 días	16-07-14	21-07-14
Diseño del sistema	4 días	16-07-14	21-07-14
Esquema prototipo del programa	2 días	18-07-14	21-07-14
Desarrollo del programa	**37 días**	**22-07-14**	**11-09-14**
Creación de esqueleto programa	4 días	22-07-14	28-07-14
Implementación clase template	31 días	28-07-14	08-09-14
Integración con clase create_exam	14 días	01-08-14	20-08-14
Integración con clase exammaker	11 días	20-08-14	04-09-14
Desarrollo clases complementarias	1 día	04-09-14	05-09-14
Pruebas unitarias	4 días	05-09-14	11-09-14
Pruebas	**6 días**	**08-09-14**	**16-09-14**
Pruebas en conjunto	6 días	08-09-14	16-09-14
Depuracion errores	6 días	08-09-14	16-09-14
Documentacion del proyecto	**8 días**	**12-09-14**	**24-09-14**
Creación del documento	8 días	12-09-14	24-09-14
Revisión final	3 días	19-09-14	24-09-14

Ilustración 22: Diagrama de Gantt de la planificación propuesta

6.2. Presupuesto

Tras haber conocido el esfuerzo resultante del desarrollo de este Trabajo de Fin de Grado, vamos a ocupar esta sección en calcular los costes totales que surgen de la consecución de este proyecto.

El coste total vamos a dividirlo en dos apartados diferenciados, el coste por el esfuerzo humano en horas destinadas al proyecto y los gastos de materiales y herramientas usadas en función de su amortización.

Comenzamos por tanto a calcular el coste personal, teniendo en cuenta las horas empleadas como variable principal de la ecuación. Para diferenciar el papel de cada una de las personas que han participado en el presente trabajo, se ha dotado de una remuneración acordada entre los siguientes roles:

- Jefe de proyecto a 55 €/hora
- Analista: 33.5€/hora
- Programador: 30.5€/hora

Una vez se define la remuneración que corresponde a cada trabajador según su rol, se presenta a continuación aquellos actores que han formado parte en este Trabajo Fin de Grado:

- Álvaro Ibáñez Aguirre: desarrollador del proyecto.
- Jesús Arias Fisteus: director del proyecto. Aportó asesoramiento y revisiones.
- Myriam Aguirre Cavero: asesor externo. Aportó asesoramiento y revisiones.

La siguiente tabla muestra el total de coste personal atendiendo a los datos antes mostrados:

Cargo	Nombre	Trabajo realizado	Coste/ hora (euros)	Tiempo empleado (horas)	Coste Total (euros)
Programador	Álvaro Ibáñez Aguirre	▪ Diseño conceptual ▪ Codificación ▪ Pruebas ▪ Redacción	30.5	336	10248
Jefe de Proyecto	Jesús Arias Fisteus	▪ Asesoramiento ▪ Revisión	55	60	3300
Analista	Myriam Aguirre Cavero	▪ Asesoramiento ▪ Revisión	33.5	30	1005
				TOTAL	**14553€**

Tabla 2: Costes de personal

Como segunda parte del cálculo de costes, vamos a estudiar los gastos materiales que este proyecto ha acarreado. Para ello vamos a emplear la siguiente fórmula basada en la amortización de equipos, la cual atiende a las variables: coste del producto, tiempo de uso, factor de utilización y la vida útil del artículo.

$$Coste = \frac{Coste\ producto(€)\ x\ Tiempo\ uso\ (meses)\ x\ Factor\ amortización\ (0.0 - 1.0)}{Vida\ útil\ (meses)}$$

Se muestra en la siguiente tabla los costes ocasionados por los materiales:

Recurso	Utilidad	Coste fijo(€)	Tiempo de uso (meses)	Amortización (0.0 – 1.0)	Vida útil (meses)	Coste total (euros)
Ordenador sobremesa i5 3570k	Ordenador principal utilizado para el desarrollo del proyecto.	900€	3	1.0	12	225
Ordenador portátil Asus Dual Core	Ordenador secundario utilizado para la fase de programación	1000€	1	0.5	58	9
Microsoft Office Professional Plus 2010	Redacción documentos y elaboración presentación	150€	1	0.8	12	10
Internet Banda Ancha	Búsqueda información y descarga del software libre necesario	50€	3	1.0	3	50
Gastos Indirectos	Relacionados con luz, agua etc.	50€	3	0.1	3	5
					Total	**299€**

Tabla 3: Costes materiales

Finalmente calculamos el coste total con los datos obtenidos anteriormente:

$$Coste\ total = Coste\ personal + Coste\ material$$

Costes personales	14553€
Costes de la amortización material	299€
Total costes	**14782€**

Tabla 4: Costes totales

7. Marcos

7.1. Entorno socioeconómico

La solución desarrollada tiene como objetivo proveer de una herramienta de uso académico que se espera tenga una alta aceptación en entornos educativos dadas sus características.

Por un lado, se ha desarrollado el *software* en código libre, lo cual permite el uso y modificación del código fuente sin restricciones más allá de las impuestas por las licencias GNU GPL bajo las que está el proyecto, que no permiten su uso para código no libre.

Por el motivo anterior, se deduce también que los gastos derivados del uso de la aplicación son nulos, incurriendo únicamente en gastos en el caso en el que se quiera contratar a un programador para modificar el código fuente, pero en ningún caso será necesario para hacer uso del *software*. Además al estar diseñado Eyegrade para usarse con cualquier webcam de gama baja, también se minimizan al máximo los posibles gastos en hardware.

En resumen, esta herramienta será muy útil en ámbitos de centros de educación especialmente las instituciones públicas de menos tamaño como los colegios e institutos, que disponen de menos presupuesto, siendo posible su expansión en otros lugares como centros privados o universidades que aun disponiendo de mayores presupuestos, puedan optar por una opción asequible y funcional como esta.

7.2. Marco regulador

Tras realizar la búsqueda de algún marco regulador que influyera en el desarrollo de este proyecto, no se ha encontrado ninguna referencia que suponga una restricción legal en el uso o desarrollo de esta herramienta.

Dependerá entonces, del grado de limitación impuesto por el centro educativo o por el propio profesor encargado de crear el documento, que la confidencialidad de la información manejada no quede expuesta. Para ello, se habrá de apelar al sentido común y nunca realizar acciones como alojar los exámenes en servidores externos si no se quiere arriesgar a comprometer el trabajo.

A pesar de que no se haya encontrado un marco regulador, que afecte a la solución desarrollada, se sabe que esta se integrará en el proyecto Eyegrade, el cual sí que hace un manejo de listado de datos personales de estudiantes y sus respectivas calificaciones.

Por este motivo y aunque no concierne a esta parte del proyecto en cuestión, se ha considerado oportuno mencionar los marcos reguladores que afectan al manejo de esa información. Se detallan a continuación:

Con anterioridad a la Ley Orgánica 15/1999 de 13 de diciembre de Protección de Datos de Carácter Personal, (LOPD) publicar en los tablones o incluso en Internet las calificaciones de los exámenes efectuados por los alumnos no suponía ningún problema, pero con la entrada en vigor de esta normativa (enero de 2000), la publicación de esta información supone una infracción muy grave al entenderse que en estos casos se está efectuando una cesión de datos personales (nombre + apellidos +nota del examen) sin el debido consentimiento del afectado, esto es, del alumno. **[10]**
La propia Agencia Española de Protección de Datos ha emitido un Informe Jurídico (el 469/2004) **[11]** informando que publicar las notas supone una cesión de datos sin consentimiento. En consecuencia, la difusión de dichas notas de calificación a través de los tablones de anuncios de la Universidad, constituirá una cesión de datos de carácter personal de los alumnos, Ley Orgánica (artículo 11.2. a), y por lo tanto sería necesario que ellos mismos dieran su consentimiento para poder realizar la publicación. Sin olvidar la obligación de notificar a los interesados las resoluciones administrativas que afecten a sus derechos e intereses se establece en artículo 58 de la Ley 30/1992, de 26 de noviembre, de Régimen Jurídico de las Administraciones Públicas y del Procedimiento Administrativo Común **[12]**. Resultando posible la publicación de los datos de carácter personal, siempre y cuando la convocatoria determine expresamente el lugar de publicación y rigiendo sobre dichos supuestos el principio de publicidad derivado de la aplicación de lo dispuesto en los mencionados preceptos.

8. Future lines of research

This section is intended to show the advances that have been considered as plausible extensions to this Final Project.

The project is based on a tool to develop multiple choice tests and export them to PDF, so the possible improvements cover a wide range of different options depending on the specific needs of each user. We are facing a program with a great potential of improvement.

The improvements proposed below have been thoroughly tested to ensure their perfect implementation on the current project. Having been developed under an open source language and extended as is Python, facilities will be provided to the future developer, as well as guidelines towards a fuller version.

They are presented on the following blocks:

8.1. Improved template

The most logical and initial improvement would be to optimize the internal program code, since despite of having made a fully functional template, the organization of the code could improve. We suggest a possible future redevelopment of the code which facilitates understanding for future users.

8.2. Use of vector graphics

It has been found useful to incorporate functionality in order to create vector graphics (under extension .eps .pdf) because ReportLab is not currently compatible with them. In the current project we have sought the solution of using PNG images thanks to the PIL library. Further enhancements would be needed on this point to give the program more strength, and thus resemble as much as possible the features offered by LaTeX.

This improvement requires an in-depth study of the image and vector usage of Python to generate a code line that can withstand ReportLab when to include it.

8.3. Text marks

The Implementation of text tags would add great value to our program. Despite the inclusion of this feature has not been critical to the final result, it will be a requirement for future users of our program.

8.4. Translation to Python version 3.x

In order to provide greater scalability to our project, our program should be adjusted to the 3.x versions of Python. This way we could ensure the persistence of code in a future where the 2.x versions of Python cease to be used, also to reach a wider range of developers to contribute with their ideas and improve the program.

9. Findings and conclusions

When we embarked on this project we knew we faced a great challenge for several reasons: to develop in a unknown language and in an unfamiliar environment, having to learn everything related to Python and its libraries, and more specifically ReportLab; on the other hand we faced the handicap of immersing into someone else's project with the ambition to develop an alternative to something that was already fully functional, with the complexity of working on an unknown source and with no capacity to receive in-depth explanations of decisions upon this design.

After completing the project, we can ensure that:

- The application development is a real and practical alternative to the existing one, providing a different approach to document layout.
- The mentioned requirements are fulfilled except for one, the implementation of language marks in the texts of questions and choices, this decision was taken because of time and not being a critical point for system operation was decided to leave as a future work line.
- The project leaves marked guidelines for future programmers leaving four possible lines of research and development that would enhance the software.

Personally, the project has helped us to acquire new skills and knowledge to develop applications in Python, and PDF layout programs.

With the above, we can conclude that the project expectations have been met in a satisfactory manner.

10. Bibliografía

[1] General Public License http://es.wikipedia.org/wiki/GNU_General_Public_License

[2] GEXCAT Gestión de Exámenes y corrección automática de test http://www.gexcat.com/

[3] Python 3.0 Release https://www.python.org/download/releases/3.0/

[4] Python - Wikipedia, la enciclopedia libre http://es.wikipedia.org/wiki/Python

[5] http://www.unocero.com/2013/04/18/python-3-3-por-encima-de-la-version-2-7

[6] LaTeX http://es.wikipedia.org/wiki/LaTeX

[7] DocBook http://es.wikipedia.org/wiki/DocBook

[8] OpenDocument http://es.wikipedia.org/wiki/OpenDocument

[9] http://www.web2pyslices.com/slice/show/1564/pdf-with-ReportLab-ii

[10] Ley Orgánica 15/1999 de 13 de diciembre de Protección de Datos de Carácter Personal, (LOPD) https://www.boe.es/buscar/doc.php?id=BOE-A-1999-23750

[11] Informe jurídico 469/2004 de la Agencia Española de Protección de Datos http://www.agpd.es/portalwebAGPD/mapasitio/index-ides-idphp.php

[12] Artículos 58 y 59 de la Ley 30/1992, de 26 de noviembre, Régimen Jurídico de las Administraciones Públicas y del Procedimiento Administrativo Común https://www.boe.es/buscar/act.php?id=BOE-A-1992-26318

González Duque, Raúl: Python para todos. URL en https://launchpadlibrarian.net/18980633/Python%20para%20todos.pdf 2014

ReportLab User Guide. URL en https://www.ReportLab.com/docs/ReportLab-userguide.pdf 2014

Programming in Python 3: A Complete Introduction to the Python Language dic. 16, 2008 ISBN: 9780137129294

What's New In Python 3. https://docs.python.org/3.1/whatsnew/3.0.html

Python 2.6 Text Processing Beginners Guide, (Inglés) 18 dic 2010

ISBN-10: 1849512124

Python 2.6 Text Processing Beginners Guide Paperback – December 14, 2010 ISBN 139781849512121

Python Pocket Reference (Pocket Reference (O'Reilly)) **ISBN**-13: 978-0596158088

Beginning Python: From Novice to Professional – October 3, 2005 ISBN13: 978-1-59059-519-

Python (programming language) http://en.wikipedia.org/wiki/Python_(programming_language)#Features_and_philosophy

Programming in Python 3: A Complete Introduction to the Python Language

by Mark Summerfield Publisher: Addison-Wesley Professional Dic. 16, 2008

ISBN: 9780137129294

LaTex http://navarroj.com/latex/ Última actualización: 20 Feb 201

Sanguino Botella, Javier. Iniciación a LaTeX2e. Un sistema para preparar documentos, Madrid, Addison-Wesley, 1997. ISBN 84-7829-013-3

El libro de LaTeX, B. Cascales, P. Lucas, J. M. Mira, A. J. Pallarés y S. Sánchez-Pedreño., Madrid, Pearson, 2003. ISBN 84-205-3779-9

Developing with PDF: Dive Into the Portable Document FormatPaperback – October 27, 2013

-El B10g de Genghis http://blog.pucp.edu.pe/item/15697/el-estandar-odf-y-el-estandar-ooxml

DOCBOOK http://es.wikipedia.org/wiki/DocBook

DocBook Xsl: The Complete Guide (4th Edition)(Inglés) – 1 sep 2007 **ISBN-13:** 978-0974152134

ODF **http://es.wikipedia.org/wiki/OpenDocument**

http://es.slideshare.net/VERITO65/ventajas-y-desventajas-de-openoffice-16180799

-OpenOffice y LibreOffice (Manuales Imprescindibles)– 21 feb 2012 de José María Delgado y Francisco Paz González- ISBN-13: 978-8441531178